

36

LA REVUE QUI NE VOUS FAIT PAS SÉCHER

MAI 1987

EDITORIAL

L'évolution du matériel dont disposent les adhérents JEDI s'oriente fermement vers la gamme 16 et 32 bits. En tête du hit-parade IBM et compatibles (clônes aux yeux bridés) talonnés de près par ATARI ST et COMMODORE AMIGA. On ne parle presque plus d'APPLE II, HECTOR HRX, THOMSON et autres familiaux. A qui la "faute" ?

Prenons le cas THOMSON: voici une machine qui aurait pu, lors de l'évolution de sa gamme, exploiter un DOS existant style Flex ou OS9 tout en accusant une baisse de prix plus prononcée. Mais l'inertie commerciale semblant être la force dominante, notre chère compagnie nationale s'évertuait à vendre au prix fort des logiciels incopiables qui étaient plus abordables sur d'autres systèmes. Résultat, exit THOMSON. Que peut-il encore attendre d'un marché qui ne l'attend pas. A moins d'une "révolution" dans le concept du micro familial, l'avenir s'annonce BOUCHE.

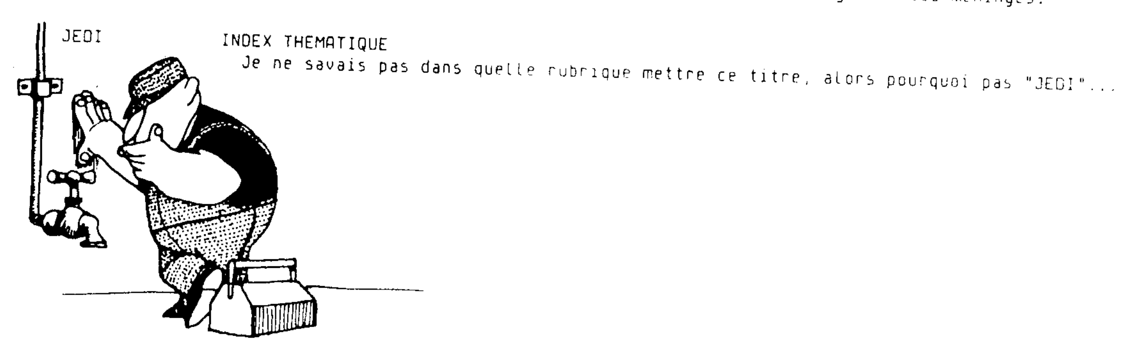
Les autres ont pris les devant. On ne compte plus les compatibles. Acquérir un compatible (pour moins de 5000 Fr maintenant), c'est se garantir l'accès à des milliers de logiciels de qualité, utilisés par des millions de personnes de part le monde. C'est disposer d'une palette impressionnante de cartes et d'accessoires éprouvés permettant d'étendre à l'infini les capacités du système.

Donc, ce qui fait la valeur d'un système ne tient pas tant à ses qualités techniques propres, ce dont pratiquement personne ne peut juger avec objectivité, mais à la qualite et au nombre de logiciels et d'accessoires disponibles. Dans ce domaine, très peu de machines peuvent prétendre à l'universalité, même dans le domaine de l'informatique lourde dont les applications restent très spécialisées.

On peut faire un parrallèle entre l'évolutiuon de la micro-informatique et l'évolution des espèces animales: mieux vaut savoir s'acclimater aux conditions les plus diverses, varier ses menus et être peu encombrant. Pourquoi croyez-vous que les rats ont survécu aux dinosaures?

Cependant, il restera toujours des systèmes marginaux, mais il ne pourront percer que s'ils savent copier les fonctions normallement traitées par la majorité des autres systèmes. Pour exemple, le NOVIX4000 semble très prometteur, mais il ne sera vraiment intéressant que quand on disposera d'un BASIC ou d'un PASCAL écrit à partir de FORTH-NOVIX4000; il est des concessions nécessaires pour survivre.

SOMMAIRE



Nos coordonnées: ASSOCIATION JEDI 17, rue de la Lancette 75012 PARIS
tel président: (1) 43.40.96.53
tel secrétaire: (1) 46.56.33.67

PRISE EN COMPTE DU CLAVIER DANS UN FICHIER BATCH SOUS MSDOS

par Jean-Marie PREMESNIL

Je connais des personnes capables d'écrire des programmes **entiers en "BATCH FILES"**, ces fichiers de commande MS/DOS **interprétés.** Il est toutefois une fonction qui, quoique **très utile, manque à cet interpréteur** : la prise en compte **d'une touche** au clavier. Car l'équivalent de KEY - **WRITE(Kbd,...) - INPUT$** - getch() (FORTH, Turbo Pascal, **BASIC, C; respectivement)**, avec ou sans écho n'a pas été **implémenté dans** l'interpréteur -génial par ailleurs-. Donc **la seule solution** laissée au programmeur est de créer des **petits fichiers** X.BAT avec X le choix de l'utilisateur.

Ces programmes, quoique de faible longueur, prennent chacun **un bloc de 512 octets**, donc sont très gourmands en mémoire **de masse.**

Pour remédier à cela deux programmes de 8 (huit) octets **chacun suffisent** à satisfaire la demande générale (ces **programmes prennent** quand même 512 octets mais ils ont le **net avantage d'être deux).**

Je les ai baptisé CODE.COM et ECODE.COM, le deuxième avec **écho. Voici les programmes:**

CODE.COM

Code objet	Assembleur
B4 00	MOV AH,00
CD 16	INT 16
B4 4C	MOV AH,4C
CD 21	INT 21

ECODE.COM

B4 01	MOV AH,01
CD 21	INT 21
B4 4C	MOV AH,4C
CD 21	INT 21

Explications : de très nombreuses fonctions du DOS sont appelés par des commandes d'interruptions, ceci est une méthode privilégiée d'appeler certaines routines très utiles. Il existe 256 possibilités d'interruption. Trois nous intéressent particulièrement.

- INT 16h Interruption de type IBM intéressant la gestion du clavier. En mettant au préalable l'accumulateur AH à 00 l'interruption arrête le programme et attend qu'une touche du clavier soit pressée. Le code ASCII de cette touche est chargée dans l'accumulateur AL et le programme continue.

- INT 21H Interruption de type DOS regroupant les principales fonctions de gestion de l'ordinateur. En mettant au préalable l'accumulateur AH à 01 l'on obtient la même fonctionnement que par l'interruption ci-dessus mais avec écho sur l'écran. Si l'on charge l'accumulateur de la valeur 4C H, on provoque l'arrêt normal du programme.

Bien. Maintenant nous allons voir comment utiliser l'un de ces deux fichiers. Donc nous avons dans l'accumulateur AL le code ASCII du caractère frappé. Or, hasard extraordinaire, il existe une fonction BAT qui justement teste l'accumulateur AL pour savoir si il y a eu une erreur transmise par le DOS. Cette commande est ERRORLEVEL dans une ligne du genre :

```
IF ERRORLEVEL 48 SUITE
```

ce qui veut dire: si l'octet de AL est supérieur à 48 c'est à dire au code ASCII 0, effectuer SUITE. J'ai bien dit supérieur et ceci implique de:

- mettre en premier le code immédiatement supérieur pour prévenir les erreurs et effectuer une boucle ou lancer un message d'erreur;

- tous les autres codes seront placés par ordre décroissant.

ERRORLEVEL sert à tester évidemment l'accumulateur pour détecter les erreurs à la suite d'une commande MS/DOS. Si la commande est exécutée avec succès, le registre AL est à 00H sinon, selon le type d'erreur, il prendra une valeur définie.

Comment faire ces programmes ? Personnellement le premier je l'ai écrit directement en... éditeur Turbo Pascal (!) en utilisant les CTRL-P suivi du caractère correspondant au code, avec une petite difficulté pour introduire le 00H (je suis sorti de l'éditeur, passé en QWERTY en pressant CTRL-ALT-F1, j'ai cherché le @, retourné en Turbo, fait CTRL-@ pour 00H, retourné sous DOS, passé de nouveau en AZERTY en pressant CTRL-ALT-F2, repassé sur Turbo, fait quelques mouvements de relaxant à mes pauvres petits doigts endoloris...). Bien entendu, un autre traitement de textes pourra faire aussi bien l'affaire (Wordstar, Edix, Personnal Editor, etc.).

Une autre possibilité est de les définir en utilisant DEBUG du DOS. Appeler DEBUG. Le "prompt" est le tiret (-). A partir de là, effectuer les commandes entre crochets et ENTER. En précisant que l'origine du programme sera à 100H [A 100], entrer tout simplement les codes assembleur tels qu'écrits ci-dessus avec un retour à chaque ligne. DEBUG traduira. Quand terminé, valider par un dernier retour chariot. Le "prompt" (mais comment peut-on traduire ce mot en français ?) revient. Puis donner le nombre d'octets du programme en appelant le registre CX [R CX] et en le forçant à 8 [8] après les deux points; ensuite nommer le programme [N X:\CODE.COM] (ou) [N X:\ECODE.COM] puis écrire le programme ainsi réalisé [W]. Pour terminer, sortir de DEBUG [Q].

Bien entendu, d'autres méthodes de mini-programmation machine sont possibles, comme pour exemple la réalisation de créateurs de logiciels en langages évolués. Je laisse au choix du lecteur l'option choisie.

CONFERENCE DU
FORTH MODIFICATION LABORATORY
information de FIG HAMBOURG

INVITATION & CALL FOR PAPER

euroFORML conference
on the FORTH programming language
and FORTH processors

from September 18th through 20th 1987
at Stettennfels Castle, Federal Republic of Germany

sponsored by
"Forth Gesellschaft eV, FRG"
and
"Forth Interest Group Inc, USA"

----

conference euroFORML
(FORTH MODIFICATION LABORATORY)
sur la programmation en langage FORTH
et les processeurs FORTH

du 18 au 20 septembre 1987
au château de Stettenfels, République Fédérale d'Allemagne

sponsorisé par
"Forth Gesellschaft eV, FRG"
et
"Forth Interest Group Inc, USA"

EuroFORML Konferenz
18 - 20 September 1987
Schloss Stettenfels, Bundesrepublik Deutschland

Veransdalter
"Forth Gesellschaft eV, FRG"
und
"Forth Interest Group Inc, USA"

Participant DM 640,--
(hébergement au château, trois repas par jours et documents de la conférence)

Invité DM 490,--
(hébergement au château, trois repas par jours)

Suite page 7

INTRODUCTION A L'INTELLIGENCE ARTIFICIELLE
par Jean-Marie PREMESNIL

INTRODUCTION

L'informatique laisse de plus en plus la place de la logique binaire et du calcul de précision au raisonnement, voir au bon sens. Voici venu le temps où l'introduction de données au clavier se fait en langage naturel, où l'ordinateur "comprend" les paroles et les manuscrits.

La nécessité d'accéder au plus grand nombre s'est fait sentir autour des années 70 lorsque les systemes se sont réduits en taille et augmentés en puissance.

Nous sommes actuellement dans une période ou les ordinateurs multiplient leur capacite par 10 et divisent leur coût par 2 tous les 10 ans. Les concepteurs de logiciels sont obligés de rendre leurs produits conviviaux en raison de la banalisation et de l'accessibilité des micro-ordinateurs.

L'intelligence artificielle désigne tous les traitements qui font penser que l'ordinateur tient des raisonnements humains.

INTELLIGENCE, DEDUCTION, LOGIQUE HISTORIQUE

SOCRATE -470--399: Syllogismes, induction, maïeutique de la pensée humaine.

Blaise PASCAL 1623-1662: Première machine arithmétique à l'âge de 19 ans. Fonde les principes du calcul des statistiques.

Charles BABBAGE 1792-1871 et Ada LOVELACE: Inventent les principes de base des ordinateurs modernes. Posent les principes des machines à calculer à quatre opérations et l'auto- programmation.

Machine de TURING 1936: Machine universelle à bandes perforées, les principes de base des processeurs de la première génération.

VON NEWMAN et SHANON: Dégagent les règles qui peuvent permettre de jouer et de gagner automatiquement à certains jeux de statégie (algorithme de Minimax).

Langage IPL 1955: Ancêtre des langages de manipulation des symboliques et non uniquement de nombres.

GENERAL PROBLEM SOLVER 1958: Système automatique de raisonnement logique avec buts et sous-buts.

1975-2000 (?): Systèmes experts, aides à la décision, langages naturels, reconnaissances diverses (formes, parole, etc.)

APPLICATIONS ET DOMAINES

JEUX:
- Echecs, jeux de rôle.

SYSTEMES EXPERTS (S.E.):
- MYCIN: système de diagnostic medical conçu en 1973, le premier S.E. reconnu.

- PROSPECTOR: système de recherche géologique et minière. Quelques centaines de générateurs de S.E.actuellement sur le marché permettent aux entreprises et aux particuliers de développer des applications sur micros et minis (Expert2 de MVP, Guru de ISE-Cegos sur micros par ex.).

La gestion technique centralisée (G.T.C.) qui consiste à grouper tous les renseignements d'un ensemble d'outils techniques pour sa gestion et sa maintenance, peut très bien être commandé par un S.E. En raison de sa souplesse de programmation et de sa facilité d'accès, cet outil remplacera dans un proche avenir les logiciels lourds et difficiles à gérer et à dépanner.

TRADUCTEURS: Lexicaux, grammaticaux, sémantiques. Nous connaissons les traductions du style mot à mot qui sont généralement en dehors de tout le contexte de l'écrit. Il a fallu des mémoires et des programmes de transcription très importants pour que les traducteurs prennent en compte la syntaxe et la grammaire des langages traduits. Malgré ses perfectionnements, il arrive encore qu'un ordinateur traduise une phrase en anglais aussi banale que "time flies like an arrow" par "les mouches à temps aiment une flèche" au lieu de "le temps vole comme une flèche". On comprend la nécessité de se pencher sur la sémantique des langages pour tenter de les adapter. Cette performance est encore à affiner.

CONCEPTION ASSISTEE PAR ORDINATEUR (C.A.O.):

Les techniques de conception actuelles sont fondées sur des connaissances approfondies dans les domaines spécifiques. On utilise de préférence des bases de données de grande capacité et des S.E. pour assister les logiciels graphiques de la C.A.O. Les utilisateurs de ces ensembles informatiques sont :
- les industries aéronautiques et spatiales,
- les industries automobiles,
- les architectes et concepteurs de bâtiments et de travaux publics,
- les sociétés de conception de microprocesseurs,
- etc.

LES LANGAGES DE CINQUIEME GENERATION

LISP (1959) est le premier langage à manipuler des listes qui ne sont pas des entités mathématiques. Ces listes sont composées d'atomes réunis par des fonctions propres au langage qui permettent de les structurer.

PROLOG (1972) est un langage déclaratif c'est à dire qu'il est capable de trouver une solution par lui-même à partir de propositions et de règles etablies au préalable.

SMALLTALK et PLASMA (1982) sont tous deux des langages d'acteurs. L'idée est de considérer les programmes comme une vaste collection d'objets qui s'envoient des messages et se répondent mutuellement.

TRAITEMENT DE BASES DE DONNEES

Nous utiliserons de plus en plus les logiciels de transmission à partir de simples minitels ou d'ordinateurs pour interroger des bases de données. Le volume grandissant des informations obligera les concepteurs de logiciels à traiter les bases de données par des programmes en intelligence artificielle. Les grands principes à mettre en place sont :

- la classification rapide des données,
- l'interconnexion des données sur leurs clés,
- la recherche "instantannée" des fiches,
- la recherche par l'utilisateur en langage naturel.

SOURCES SCIENTIFIQUES

L'intelligence artificielle, ou plutôt le raisonnement artificiel trouve ses sources dans les mathématiques dites modernes. C'est dire que ses principes ne sont connus que de fraîche date.

Algèbre de Boole: Ces principes ont amené directement les bases de la logique qui sont essentiels aux calculs informatiques puisque tout processeur est dérivé d'une logique binaire.

Théorie des Ensembles: Les principes amenés par les recherches de Galois ont permis un développement considérable notamment dans la théorie des graphes et par là, de la recherche opérationnelle d'une part et des sous-ensembles flous d'autre part qui sont les deux principes essentiels de l'analyse actuelle en intelligence artificielle. Les recherches actuelles sur l'intelligence artificielle sont paradoxalement axées sur les domaines de la psychanalyse et de la médecine du cerveau qui essaient de trouver les mécanismes de la mémoire immédiate et du raisonnement syllogique. Ces recherches portent sur :

- les phenomènes de "trous de mémoire",

- la mémoire associative.

APPORTS

Les domaines les plus prisés par l'intelligence artificielle se trouvent dans :

- la recherche scientifique (médecine, géologie, recherche pétrolière, recherche fondamentale).
- les entreprises gestionnaires (banque, assurance, administration),
- les entreprises de haute technologie (aviation, chimie, mécanique de précision, ingénierie).

Les banques de données vont prendre dans un futur proche une part importante dans la recherche des S.E. du fait de l'accroissement des informations à traiter et de la complexité du système connectif.

La reconnaissance des formes et de la parole peut être le début de l'ordinateur "intelligent". Nous pouvons supposer qu'il commandera des robots ou des machines outils au seul son de la voix. Toutefois, ce domaine sera réservé à des applications particulières telles que la chirurgie, la prothétique, l'armée.

SYSTEMES D'INTELLIGENCE ARTIFICIELLE

JEUX Cette application n'est interessante que par les apports mathématiques qu'elle engendre. Le jeu d'échecs est le domaine qui a le plus contribué au développement scientifique de l'intelligence artificielle grâce à la simplicité de ses règles et à la complexité des phases de jeu :
- surface délimitée facilement repérable,
- les pièces caractéristiques à déplacements simples,
- les phases de jeu à accroissement exponentiels au fur et à mesure de l'avancement de la partie.

SYSTEMES EXPERTS

INTRODUCTION Le sujet d'étude des S.E. concerne toutes les activités de l'homme pour lesquelles aucune méthode mathématique ne peut être appliquée. Ce sont des programmes ayant pour objectif de fournir dans un domaine particulier les mêmes résultats que l'homme expert en ce domaine.

Leur création provient du constat d'échec des programmes généraux d'intelligence artificielle. Dans la programmation classique, les auteurs doivent prévoir et développer tous les cas de figure. Les problèmes posés peuvent aboutir à une combinatoire telle que ces logiciels deviendraient monstrueux. Exemple :

Pour indiquer à un médecin un diagnostic lorsqu'il fournit les symptômes, nous pourrions :
- construire une structure de données contenant toutes les maladies et leurs symptômes, - demander au médecin de taper les symptômes, - puis parcourir notre structure jusqu'à ce que nous trouvions la maladie correspondante.

La dimension de l'espace de recherche est telle qu'un programme semblable est irréalisable.

Les S.E. tentent d'aborder les problèmes d'une autre façon: ils dégagent le programmeur de cette tâche de gestion en simplifiant la mise en place du système. L'utilisateur d'un générateur de S.E. n'exprime que des fragments de connaissance, des réponses types à des situations élémentaires. Il a l'ambition de se spécialiser dans un domaine restreint de la connaissance et raisonner comme un expert humain.

LEUR FONCTIONNEMENT

Un générateur de S.E. est composé de :

- une base de connaissances faite de règles simples qui sont appelées assertion. Ces règles sont entrées par l'homme expert (le cogniticien). Par principe, ces règles ressemblent à des syllogismes socratiques et aboutissent à des "conclusions" qui sont en réalité des hypothèses de départ.
- une base de faits établie par le candide. Ces faits devront être en relation directe avec la base de connaissances, c'est à dire qu'ils reprennent des assertions connues de la base de connaissance sans pour autant que l'hypothèse de départ soit perçue par l'utilisateur.

- un moteur d'inférence, logiciel programmé pour effectuer la démonstration automatique d'une hypothèse de départ prise dans la base de connaissance à partir de la base de faits. Ce logiciel peut travailler de trois façons différentes:

1) chainage avant: cette méthode de travail consiste à effectuer une recherche systématique dans toute la base de connaissances des règles se référant à la base de faits afin d'en dégager les hypothèses de départ.

2) chainage arrière: à l'inverse, ce procédé interroge l'utilisateur sur les règles les plus en amont pour aboutir logiquement à l'hypothèse de départ. Cette méthode de travail est beaucoup plus rapide et moins gourmande en mémoires que le chainage avant. Toutefois, la base de connaissance doit être structurée plus sévèrement.

3) chaînage latéral: combinaison des chainages avant et arrière. Cette méthode suppose un algorithme utilisant la pré-équivalence (voir annexe 1) pour calculer les chances les plus grandes d'aboutir à une hypothèse de départ. La base de faits n'est prise en compte que si le logiciel ne peut trouver par lui-même l'hypothèse de départ.

LA PROGRAMMATION

La programmation dans un S.E. n'est qu'une suite de règles proposées à l'utilisateur dans un ordre quelconque en apparence. En fait, il faut penser à sérier les règles par affinité afin de réduire les temps d'accès au minimum, de prévenir toutes répétitions inutiles et d'éviter les boucles infinies.

Nous allons essayer de donner une méthodologie de programmation des S.E. D'abord un exemple de base de connaissance:

"Si la terre est sèche et que je n'arrose pas, les plantes vont flétrir. Mon jardin ne sera pas beau."

Nous avons dans cet exemple :

- 2 assertions : la terre est sèche j'arrose
- 1 conclusion : les plantes vont flétrir
- 1 hypothèse : mon jardin ne sera pas beau
- 4 opérateurs logiques : SI, ET, ALORS, NON.

Ecrivons directement les règles dégagées:

(règle 1)
```
SI la terre est sèche
ET SI NON j'arrose
ALORS les plantes vont flétrir
```

(règle 2)
```
SI les plantes vont flétrir
ALORS HYPOTHESE mon jardin ne sera pas beau
```

(règle 3)
```
SI NON les plantes vont flétrir
ALORS HYPOTHESE mon jardin sera beau
```

A noter 2 points importants dans ce programme :

- l'hypothèse de la règle 3 est implicitement contenue dans la phrase de départ, et cette hypothèse est vraie dans tous les cas:

- si la règle 1 ne prouve pas l'hypothèse de la règle 2, elle ne prouvera pas son contraire. L'opération ALORS est la relation d'implication. Le diagramme de Karnaugh de cette relation est le suivant :

A	B	A=>B
1	1	1
1	0	0
0	1	?
0	0	?

C'est pourquoi la regle 3 a été ajoutee expressement. Si cette regle n'existait pas, le système aurait repondu "je ne sais pas conclure" ou un message equivalent. L'analyse d'un probleme est essentielle, nous l'avons vu avec ce petit programme. Nous allons définir quelques regles simples pour programmer un S.E:

- Partir de toutes les hypothèses de départ connues.
- Faire un tableau général de toutes les assertions aboutissant à chaque hypothese.
- Sérier les assertions en groupes généraux.
- Scinder les hypotheses en parties logiques selon les groupes généraux.
- Former chaque règle dans cet ordre sans oublier de commenter abondamment.

TYPES DE SYSTEMES EXPERTS

Les S.E. les plus simples (que l'on peut aisémment trouver dans de nombreux ouvrages) ont un moteur d'inférence d'ordre zéro à chainage avant. Ils savent résoudre des syllogismes ou des compositions de fonction à une ou plusieurs règles.

D'autres S.E. plus évolues ont un moteur d'inférence d'ordre 1 à chaînage avant et arrière et à facteurs de crédibilité. Ils peuvent traiter une centaine de règles et sont très utiles pour accéder à la programmation et pour faire des experts simples mais performants. Ce type de logiciel, édité par les principales maisons de software, fonctionne sur micro-ordinateurs compatibles IBM PC.

Les S.E. très évolués peuvent avoir plusieurs moteurs d'inférence parallèles et fonctionnent sur des bases de données relationnelles. Ils sont très lourds et très difficiles à programmer. Ces S.E. sont spécialisés dans des domaines particuliers comme la médecine ou la recherche minière.

LES TRADUCTEURS

INTRODUCTION:
La traduction automatique des textes a été l'une des recherches les plus assidues de l'intelligence artificielle, mais les résultats ont été moins rapides que ne le pensaient les premiers chercheurs. Au début, il a fallu essayer de réglementer très strictement la grammaire, et ce sous forme d'entités mathématiques. Des 1949, l'informaticien Warren Weaner tente la traduction d'un texte par un ordinateur. Il s'aperçut bien vite que l'analyse lexicale arrivait à des aberrations telles que celle-ci : "the spirit is willing but the flesh is weak" qui se traduit par "l'esprit est consentant mais la chair est faible" devient par la traduction lexicale "l'alcool est prêt mais la viande est avariée". Les textes ne sont donc analysés que sous leurs deux aspects : syntaxique et sémantique.

FONCTIONNEMENT:
Le grand progrès de la grammaire moderne découle de l'analyse informatique. La grammaire en constituants immédiats structure une phrase selon un arbre comme défini ci-dessous :

```
P ──────> GV ──────> verbe
   └────> GN
|
├──────> GN ──────> adjectif
├──────> article
├──────> nom
|
└──────> PP ──────> GN
```

avec :
P = la phrase
GN = le groupe nominal
GV = le groupe verbal
PP = la preposition

Ceci constitue l'arbre syntaxique compris facilement par un logiciel. Le texte de départ est éclaté en un arbre qui sert de génération au code machine. Le logiciel tente de réécrire le texte dans la langue desirée en suivant la syntaxe appropriée. Si cette traduction échoue le logiciel revient en arrière pour changer la structure de l'arbre pour faire correspondre les syntaxes des deux langages.

Actuellement, un logiciel de traduction syntaxique se présente comme un ensemble de logiciels indépendants qui s'occupent chacun d'une partie spécifique de la syntaxe et qui sont dirigés par un programme maître. Parallèlement, les informaticiens ont essayés de prendre une phrase non pas comme une entité mathématique mais sémantique, dans le but d'avoir une approche plus naturelle de la pensée humaine.

Les hypothèses de division des phrases par cas ont été nombreuses. Pour exemple : un groupe de mots peut-être un instrument, un lieu, une durée, un sujet, un objet, un but, une source, etc. Une phrase peut être syntaxiquement égale à une autre mais être sémantiquement totalement différente :

"Pierre mange une glace à la fraise"
et "Pierre mange une glace à la maison".

La traduction dans une autre langue de ces deux phrases peut avoir une syntaxe différente.

On utilise, dans ces logiciels des "démons" qui repèrent des groupes de mots significatifs (proverbes, sentences, lieux communs, idiotismes) et qui les traduisent systématiquement.

Tout ceci est bien joli mais ces traducteurs ne peuvent actuellement s'occuper que de textes simples et de portée générale. On s'imagine mal la taille en mémoire considerable que doit avoir un système de traduction automatique pour s'occuper d'une seule phrase de Marcel Proust.

LES APPLICATIONS

Les travaux de linguistique appliqués à l'informatique dévient vers des logiciels de construction de récits. Les écoliers, premiers bénéficiaires, commencent déjà à avoir des programmes d'élaboration d'histoires sur des thèmes programmés semblables aux jeux de rôles; les caractères des personnages, les rapports entre eux, les situations, les aventures sont enregistrés et peuvent être modifiés par l'imaginaire des enfants.

LA CAO:

INTRODUCTION:
Auparavant, ce domaine de programmation ne faisait pas partie de la famille de l'intelligence artificielle. Maintenant ces vastes logiciels intègrent aussi bien des S.E. que de la reconnaissance des formes, ou bien des générateurs automatisés.

Les programmes de conception assistee par ordinateur sont de lourds ensembles comprenant :

- un logiciel de dessin en deux ou trois dimensions,
- un calculateur interface avec les fichiers de dessin,
- des optimiseurs de formes, de dispositions, de volumes,
- quelquefois des commandes d'automates pour la réalisation des prototypes.

Ces ensembles tournent sur de gros systèmes (VAX, IBM 5080) dotes de memoires allant de 10 Mo à plusieurs Go. Leurs programmations sont très delicates et toujours adaptes aux entreprises qui utilisent ces systemes.

LES TYPES DE CAO:
Les C.A.O. qui peuvent être intégrées a des systemes type IBM PC sont destinees à l'architecture, au dessin technique et quelquefois à la création artistique. Ces logiciels (CONCEPTION 3D, MAC SPACE sur MACINTOSCH, VERSA CAD) peuvent être couples à d'autres programmes de calculs ou d'optimisation.

LE TRAITEMENT DES BASES DE DONNEES

Les développeurs de S.E. se sont toujours penchés sur la

prise en compte des bases de données dans leur système. La préoccupation majeure était d'alimenter les bases de connaissances d'un système par les bases de données. Ceux-ci peuvent être progammés facilement et contenir de plus vastes informations notamment les relations entre les données. Les premiers S.E. étaient difficilement interfaçables avec les bases de données en raison de l'incompatibilité des langages de programmation. Ces difficultés ont été surmontés. La tendance actuelle est de prévoir des transmissions à distance d'un S.E. sur une base de données pour disposer de connaissances accessibles à tous.

On peut également combiner la technique des S.E. avec celle des bases de données soit pour établir une interface entre le S.E. et la base de données, soit à partir de la S.E. construire une base de données, soit concevoir à partir de ces deux systèmes un nouveau système de connaissances.

Nous n'en sommes encore qu'au stade des théories mais la recherche actuelle est de concevoir un système de métaconnaissances par le principe de boot-strap c'est à dire un autogénérateur de S.E. à partir de ses propres connaissances. Pour celà il est indispensable d'utiliser comme support de la base des connaissances une base de données qui pourra contenir toutes les connaissances nécessaires à sa régénération.

CONCEPTION DE L'INTELLIGENCE ARTIFICIELLE

LANGAGES DE L'INTELLIGENCE ARTIFICIELLE,
LANGAGES DE PREMIERE GENERATION:
Les premiers langages de l'informatique étaient une sucession de 1 et de 0 programmés à partir de boutons-poussoirs ou d'interrupteurs commandés par d'habiles manipulateurs. Les programmeurs devaient transformer chaque commande par le code binaire compris par la machine. Les instructions étaient à l'époque très limitées (adressages directs, sauts conditionnels en nombres restreints). Les programmes de ces années 50 (l'age de pierre de l'informatique) se contentaient de calculs numériques complexes et pouvaient même effectuer, par des algorithmes appropriés, des calculs differentiels.

LANGAGES DE DEUXIEME GENERATION
Le langage assembleur a grandement facilité la tâche de programmation. Cela suppose déjà un programme intégré au système qui traduit chaque primitive dans son code binaire. Ce langage est encore utilisé de nos jours. Les années 55 à 62 ont été florissantes en recherche d'algorithmes performants qui sont toujours valables, que ce soit dans le domaine du calcul ou bien du tri, ou encore du traitement d'informations élémentaires (CORDIC, HUFFMAN, MESNER, VON NEWMAN).

C'est à cette epoque que les chercheurs commencent à intégrer des logiciels à leur machine pour faciliter la tâche des programmeurs. Ce sont les fameux compilateurs qui à l'instar de l'assembleur, traduisent les commandes en codes binaires. La différence est que chaque commande peut engendrer un bloc de codes qui est en soi un mini programme.

LANGAGES DE TROISIEME GENERATION:
Les années 62 ont vu naitre de beaux langages encore réputés:

- FORTRAN : langage prisé des mathématiciens, toujours en vigueur, constamment actualisé. FORTRAN s'utilise principalement dans les universités et écoles à orientation mathématique, technique et scientifique.
- COBOL : comme FORTRAN, toujours en vigueur, langage de prédilection des gestionnaires et des comptables. Utilisé encore dans les grandes administrations.
- PL1 : compromis entre FORTRAN et COBOL. Tombé en désuétude.
- BASIC : le langage le plus pratique au monde. Inventé par le mathématicien d'Einstein, qui ne connaissant rien en informatique a créé un langage simple d'utilisation à partir du FORTRAN.
- LISP : le premier langage de l'intelligence artificielle. Les premiers S.E. ont été écrit dans ce langage. Toujours en vigueur et opérationnel:
- ALGOL : un des langages les plus répandus dans les universités, il est malheureusement tombé dans l'oubli. C'est peut être grâce à lui que l'informatique est devenue ce qu'elle est.

LANGAGES DE QUATRIEME GENERATION:
L'évolution de la programmation a demandé dans les années 72 des langages structurés. Ces années ont vu l'avènement de plusieurs langages, certains dérivés des précédents, d'autres entièrement repensés pour des applications particulières. La majorité de ces langages ont été conçus et implémentés par un seul homme. La philosophie et la structure de ces langages sont imprégnés de l'esprit de leur concepteur.

- FORTH (68) : Charles MOORE (USA): Ce langage a été utilisé pour la première fois sur le radiotéléscope de Kitt Peak USA. En 1976 FORTH fut adopté comme standard de programmation pour l'astronomie. Une réunion annuelle se tient aux USA pour maintenir le standard de ce langage à la pointe des recherches informatiques. Actuellement, est disponible un micro-processeur programmable directement en FORTH. Ses performances sont remarquables.
- LOGO (70) : S. PAPERT (USA): Ce langage est directement tiré du LISP, a été adapté pour faciliter les graphiques simples sur l'écran. De plus, il peut manipuler des listes complexes d'objets qui le rend particulièrement apte à la programmation en I.A.
- PASCAL (72) : WIRTH (ZURICH) Par un souci de programmation structurée et en réaction aux "bidouillages" géniaux des informaticiens de l'époque de la deuxième génération, ce langage a été créé en vue de l'élaboration de programmes et d'algorithmes strucurés par le concept de la méthode descendante. Les universitaires ont été unanimes à adopter ce langage. -
- PROLOG (72) : A. COLMERAUER (MARSEILLE): Langage déclaratif conçu par un groupe de recherche en Intelligence Artificielle. Ce langage ne se veut pas rapide en exécution mais rapide en programmation. Il est régi par un moteur d'inférence à chainage arrière qui peut le faire ressembler à un S.E.
- Langage C (1973) : THOMPSON, RITCHIE (USA): Autour du système UNIX, un groupe de chercheurs a créé un langage polyvalent et tout son environnement. Le Langage C gagne du terrain par rapport aux autres langages. Il est à la base de nombreux systèmes d'exploitation et logiciels connus (Unix, DBase III, Multiplan III, FrameWork, etc).
- ADA (1973) : (USA): langage composite, élaboré en plusieurs étapes et par plusieurs commissions, on peut dire qu'il ne reflète pas la simplicité. Le cahier des charges trop rigide de ce langage ne permet pas de l'implanter sur n'importe quel ordinateur. Les seuls utilisateurs connus sont l'armée américaine qui tente d'en faire un standard.

LA CINQUIEME GENERATION: LES NOUVEAUX LANGAGES
- SIMULA: Ce langage descend directement de l'ALGOL (comme le Pascal). Il sert à créer des objets à partir de classes spécifiques et à simuler des interactions entre ceux-ci.
- MODULA-2: Issu de Pascal, et comme celui-ci, langage procédural, MODULA-2 intègre des fichiers comme s'il s'agissait de fonctions et permet ainsi la multi-programmation,sur des systèmes multi-processeurs.
- PLASMA: Langage mixte, les procédures définissent les "acteurs" et les déclarations manipulent et font réagir ces acteurs les uns par rapport aux autres.
- SMALLTALK: Comme PLASMA, mais plus répandu car déjà implanté sur certains systèmes (MACINTOSH, GEM, WINDOW), ce langage définit un environnement bien particulier : écran haute résolution, souris, imprimante laser.Ces quatre langages ont un point commun : leur double niveau d'accès. Le premier niveau définit des modules indépendants et leur réactions lorsqu'ils sont confrontés à d'autres modules. Le deuxième niveau, plus convivial, permet de combiner ces modules pour répondre à une situation donnée.
- LE SYSTEME JAPONAIS En 1980, le Ministère japonais de l'Industrie et du Commerce Extérieur (MITI) avec 8 groupes indutriels entament un programme dit de 5ème génération qui devra se développer en 10 ans. L'objectif est de créer un ordinateur capable de résoudre les problèmes rencontrés en I.A. avec un budget de près d'un millard de dollars. Malgré les énormes moyens mis en oeuvre (laboratoires, recherches formelles, produits technologiques nouveaux) l'objectif initial semble être compromis. Le langage adopté par ce groupe est le PROLOG. Les techniques qui progressivement ont abouti au niveau actuel n'ont pas été prises en compte en phase initiale de ce projet. Petit à petit, la dérive du programme de

départ s'est orientée vers des systèmes souples et maléables. L'ordinateur intelligent espéré en 1980 s'est transformé en de multiples unités indépendantes. Leurs interconnections posent des difficultés aux constructeurs.

- LE SYSTEME AMERICAIN:

Dans la même optique que les japonnais, les U.S.A. ont voulu développer une stratégie dans la bataille de la 5ème génération. Les constructeurs privés, contrairement à l'autre camp n'ont pas suivi les directives de l'administration. Ces groupes préféraient suivre leur propre voie en toute discrétion pour se prémunir contre un éventuel échec ou à l'inverse se réserver l'exclusivité d'un succès. Seuls les grands groupes administratifs ont accepté ce projet (NASA, Pentagone, CIA, Universités). Le langage adopté est l'ADA. Les résultats de ces recherches ne seront pas connus de sitôt du grand public, ces travaux sont pour la plupart destinés à un usage militaire.

- LE SYSTEME EUROPEEN:

L'Europe aussi a son programme de 5ème génération. Il s'appelle "ESPRIT" (European Strategic Program of Research in Information Technologies). 12 grandes entreprises du secteur électroniques ont répondu à l'appel de la Communauté Européenne : Philips, Nixdorf, Siemens, Thomson, Téléfunken, ICL, Olivetti, Stet, GEC, CGE, AEG, Plessey.

Les 2 principes fondamentaux du programme européen sont:

- les recherches effectuées doivent être l'oeuvre conjointe d'industriels issus d'au moins deux états de la Communauté,
- les projets sélectionnés doivent se situés en aval de la recherche fondamentale et en amont de la réalisation concrète du produit. Conjointement, les Universités spécialisées en I.A. se sont associés aux entreprises pour apporter leur savoir en recherche fondamentale.

L'AN 2001:

Les technologies qui apparaissent nous promettent, par leur capacité et leur rapidité, des machines axées sur trois grands principes : l'intelligence, la convivialité et la fiabilité. Le stockage et l'usage de l'information et de la connaissance sera effectué par des langages naturels, accessible au plus grand nombre, interconnecté à des bases de connaissances et de données facilement exploitables.

Même si les trois grands projets de l'I.A. ne semblent pas répondre à l'attente espérée, les retombées des études faites ne pourront être que bénéfiques à l'ensemble de la communauté informatique.

BIBLIOGRAPHIE

REVUES:

MICRO SYSTEMES - Société Parisienne d'Editions (mensuel) 43, rue de Dunkerque 75010 PARIS

JEDI - Association JEDI (mensuel) 17, rue de la Lancette 75012 PARIS

Dr DOBBS'S JOURNAL (mensuel) P.O. Box 27809, San Diego, CA 92128

01 INFORMATIQUE - Groupe Tests (hebdomadaire) 5, Place du Colonnel Fabien 75491 PARIS CEDEX 10

LIVRES:

L'HOMME FACE A L'INTELLIGENCE ARTIFICIELLE J.D.WARNIER Les éditions d'organisation

SYSTEMES EXPERTS. METHODES ET OUTILS par CHATAIN DUSSAUCHOY Eyrolles

MOTEURS DES SYSTEMES EXPERTS R.VOYER Eyrolles

GODEL ESCHER BACH (Les brins d'une guirlande éternelle) D. HOFSTADTER InterEditions

MANIFESTATION:

Les journées d'Avignon Avril

Actes d'Avignon Agence de l'Informatique

■

Suite de la page 2

Participant DM 490,--
(hébergement hors du château, trois repas par jours et documents de la conférence)

Etudiant DM 320,--
(hébergement au du château, trois repas par jours et documents de la conférence)

EuroFORML est un meeting international réunissant des pratiquants de l'informatique utilisant FORTH en tant qu'outil pour résoudre leurs problemes. Il est prévu de faire lecture, démonstration et exposition des techniques utilisées.

Cette année, le thème de la conférence sera axé sur le matériel et les possibilités offertes par le saut qualitatif que représentent les nouveaux processeurs FORTH.

La conférence aura lieu au château de Stettenfels (12ème siècle) aux environs de Hilbronn près de Stuttgart, RFA. Le château peut accueillir 60 invités et peut réunir 110 participants pour les conférences. La réservation hotelière située à proximité du chateau peut être arrangee.

La langue utilisée lors des conférences sera l anglais et le FORTH, bien entendu. La conférence est auto-organisée, c'est à dire qu'il n'y a pas d'agenda des priorités pour débuter la réunion. Si vous avez une idée à présenter, choisissez le format suivant:

papier de présentation
il doit représenter 10 minutes de temps de parole en exposant au groupe les possibilités et retombées immédiates.

schéma de présentation
il vous sera assigné un emplacement de présentation des schémas éventuels à partir duquel vous pourrez présenterer votre idée à un petit groupe de personnes dans des pièces séparées. Ceci est particulièrement pratique pour les démonstrations de matériel et de logiciel.

boutiques/stands
vous pouvez organiser ou participer à des boutiques en début de conférence et en fonction de la demande.

Un document en langue anglaise sera publié après la conférence; les papiers inclus à ce document sont issus du travail des participants et reproduits par photocopie en début de conférence.

Les inscriptions sont closes début août. Un acompte de 200,-- DM par personne est requis (paiement en Deutsch Mark -faut vous y faire, il n'y a pas encore d'écus-, eurochèques en monnaie étrangère ou transfert de fond sur "Postgiro compte n° 5632 11 - 208, code bancaire 200 100 20). Le reste de la somme sera demandé en début de conférence. La place est limitée, les premiers arrivés seront les premiers servis.

Cette année, nous réservons un tiers de la place aux étudiants. Les invités risquent donc de prendre leurs repas au château, mais de ne pouvoir être hébergés. Il est possible de venir avec son matériel de camping, la pelouse du château étant assez confortable.

Pours les auteurs

Les papiers présentés pour la conférence (qui seront inclus au document) doivent être envoyés au "Forth Gesellschaft eV" avant le 1er septembre 1987. Le format est A4 avec une marge de 2,5 cm tout autour. Chaque page sera numérotée et porter le nom de l'auteur. Les documents n'excèderont pas 15 pages.

Pour réserver et envoyer les papiers de conférence, écrire ou appeler:

C.D. OSTEN
Gneisenaustr. 23 / D-2000 HAMBURG 20 / (BRD-RFA)
19-49 40 422 1694 ou 19-49 40 490 5195
(numeros valables depuis la France métropolitaine)

■

FBASEII sous MS-DOS/F83.

JEDI a publié dans son numéro 22 (Janvier 86), un programme de gestionnaire de fichiers nommé FBASE. Une version rénovée a paru dans FORTH DIMENSIONS, volume VIII/4 de Nov/Dec86, auteur E. Petsche. Le programme proposé ici en est une traduction/adaptation (écrans 1 à 20), augmentée d'une partie 'MENU' afin d'en rendre l'utilisation plus 'conviviale' (écrans 21 à 33).

Ce programme permet à l'utilisateur de définir et d'initialiser un fichier, d'entrer des données, d'interroger un fichier sur une combinaison quelconque de champs, d'effacer des enregistrements et de modifier les valeurs de champs des enregistrements.

FICHIER est le mot de définition des fichiers. Le champ paramètre (le PFA) d'un mot défini par FICHIER contient :

au déplacement 0 , le numéro du premier écran du fichier ;
" 2 , le nombre maximal d'enregistrements du fichier ;
" 4 , le nombre d'octets par écran (bloc) ;
" 6 , la taille d'un enregistrement, en octets ;
" 8 , le numéro d'enregistrement courant ;
" 10 , l'adresse de la liste des champs de ce fichier.

CHAMP est le mot de définition des champs. Le PFA d'un mot défini par CHAMP contient :

au déplacement 0 , la largeur du champ ;
" 2 , le déplacement (offset) par rapport au début de l'enregistrement ;
" 4 , le type du champ.

Le premier enregistrement de chaque fichier (0 RECORD) contient des informations concernant la longueur du fichier (LASTREC) et le nombre d'enregistrements actifs dans le fichier (#ACTIVE), ces informations occupant les 4 premiers octets de cet enregistrement.

L'écran 20 donne les définitions de FICHIER et de CHAMP pour une application: ici, un index des numéros parus de JEDI. Trois paramètres doivent être spécifiés lors de la définition d'un fichier: le bloc de début (écr. 40), le nombre maximal d'enregistrements (500 : soyons optimistes), et la longueur d'un enregistrement (100 octets).

De même trois paramètres doivent être spécifiés lors de la définition d'un champ : son type, son déplacement et sa largeur (cf écr. 20). Cette dernière doit être donnée même pour les types 'numériques', en vue de leur affichage.

CHAMPS compile la liste des CFA des mots de champs, l'adresse de début de cette liste est placée dans FIELD-LIST. Syntaxe : <nom_du_fichier> N CHAMPS suivi des noms des champs. cf. écr. 18 et 20.

Lors de la première compilation du programme, du moins celle de l'écran d'application, il est impératif d'exécuter NEWFILE <nom_du_fichier>, qui initialise les informations de l'enregistrement 0.

Dans cette version, le programme est lancé automatiquement, et vous place dans le MENU. Il utilise le mot $EXEC, qui lance une commande FORTH lorsque celle-ci est placée dans une chaîne 'comptée'. Ce mot est repris de JEDI n°33, 'Trois routines' (rendons à Marc ...).

La fonction 1 du MENU, 'Ouverture d'un fichier', n'est justifiée que si plusieurs fichiers sont installés avec le programme de base: c'était le cas ici, où 3 revues cohabitaient; il faut bien entendu leur faire des écrans d'application (tel que 20) distincts, et surtout leur donner des noms de champs tous différents.

Deux modes de recherche sont possibles: 'Recherche Normale', fonction 2 (STEP) et 'Recherche Sélective', fonction 3 (SELECT). Le premier affiche l'un après l'autre tous les enregistrements trouvés correspondant aux clefs spécifiées. Ce mode permet aussi la correction ou la suppression d'un enregistrement. Le second n'affiche que les champs désignés, toujours selon les clefs spécifiées. Le nombre "d'extraits" possibles est arbitrairement fixé à 5 (écr.18): en fait, la limite dépend de la longueur des champs sélectionnés, leur affichage doit tenir sur une seule ligne d'écran.

La recherche se fait sur une quelconque combinaison de champs pour chaque enregistrement d'un fichier. Les conditions sont SUPRA, INFRA, EST et NESTPAS, auxquelles s'ajoutent les opérateurs logiques ET et OU. Le nombre maximal de conditions (Q#) est fixé à 4, ce qui semble raisonnable car TIB n'accepte que 80 caractères.

CHERCHE est le mot d'interrogation 'utilisateur'. Il est implicite, suivi du nom du fichier, dans les fonctions de recherche du MENU. La commande est placée dans TIB, et le programme exécute le mot suivant, qui est un nom de fichier, et en fait le fichier courant. Puis le nombre de mots suivant le nom du fichier est compté (#ARGS), puis incrémenté de 1. Si ce nombre divisé par 4 donne un reste nul, le nombre d'arguments est valide, et le quotient est le nombre d'arguments pour cette recherche: il est utilisé par FOUND? et Q-ARRAYS. Les arguments de recherche (les valeurs à comparer aux champs spécifiés) sont placés dans TARGETS. Le nombre maximal d'arguments de recherche est 30. Les chaînes d'arguments numériques sont converties par NOMBRE avant d'être envoyées par BRING dans TARGETS.

Note: le programme original utilisait le mot NUMBER du noyau, lequel en cas d'erreur exécute ABORT. Pour éviter une sortie intempestive du MENU, il faut donc redéfinir ce mot. De même BOOT a remplacé ABORT" dans certains mots, pour cette raison.

Le fichier est alors examiné, en vérifiant d'abord que l'enregistrement est actif (REMOVED?), alors les arguments de recherche sont exécutés par FOUND?. Lorsque toutes les conditions ont été vérifiées, un drapeau est empilé. S'il est vrai, toutes les conditions sont vérifiées pour l'enregistrement courant, il est alors affiché.

Un mot d'affichage de la totalité du fichier n'a pas été inclus: il suffit de faire par exemple

SUJET NESTPAS XXX ET NUMERO SUPRA 25 pour visualiser tous les enregistrements correspondants.

GLOSSAIRE FBASEII.

'OPEN	variable contenant le PFA du fichier courant.
'FIELD	variable contenant le PFA du champ courant.
FIELD-LIST	adresse dans le PFA du fichier courant contenant l'adresse de sa liste des champs.
LASTREC	1er oct. de l'enregistrement 0, contient le numéro du dernier enregistrement.
#ACTIVE	3ème oct. de l'enregistrement 0, contient le nombre d'enregistrements actifs.
FICHIER	mot de définition d'un fichier. Lorsqu'un mot défini par FICHIER est exécuté, il place son PFA dans 'OPEN.
CHAMP	mot de définition d'un champ. Lorsqu'un mot défini par CHAMP est exécuté, il place son PFA dans 'FIELD et empile l'adresse du champ.
FLD-WIDTH	contient la longueur du champ courant.
FLD-TYPE	les types de champ sont ALPHA (codé 0, pour du texte), SIMPLE (codé 2, nombre simple), DOUBLE (codé 4, nombre double) et $$ (codé 6, pour un affichage en 'dollars').
TABLE	mot de définition pour les tables d'exécution des fonctions dépendant du type. Lorsqu'un mot défini par TABLE est exécuté, il utilise la largeur du champ courant pour sélectionner la fonction devant être exécutée.
(ENTER)	table d'exécution contenant les mots d'entrée de tous les types de champs. A l'exécution, les mots de cette table attendent une adresse de champ sur la pile.
DISPLAY	table d'exécution contenant les mots d'affichage de tous les types de champs. Une adresse de champ doit être empilée avant l'exécution.
COMPARE	table d'exécution contenant les mots comparant les champs aux arguments de recherche. Ces mots attendent 2 adresses sur la pile et retournent -1, 0 ou 1 pour <, = ou >.
ENTER	demande à l'utilisateur l'entrée du contenu d'un champ, et la place dans le fichier.
REMOVED?	empile un drapeau vrai si l'enregistrement courant a été marqué comme effacé.
#ARGS	compte le nombre d'arguments restant dans TIB. Ce mot devra être modifié si les commandes se font par chargement d'un bloc.
Q#	nombre maximal de conditions de recherche.
#HITS	nombre d'enregistrements trouvés lors de la recherche. Utilisé comme drapeau, et par SCRFUL? pour arrêt de l'affichage sur 'écran plein'.
LOGICALS	tableau des opérateurs logiques (ET et OU) devant être exécutés par la recherche.
OPERANDS	tableau des opérandes de champs devant ête comparés lors de la recherche.
CONDITIONS	tableau des conditions de recherche (SUPRA, INFRA, EST, NESTPAS).
TARGETS	adresse de la zone de sauvegarde des arguments de recherche, ici en HERE+200.
+TARGET	utilise l'indice empilé comme déplacement dans TARGETS. Synonyme: TARGET.
BRING	table d'exécution pour les mots amenant les arguments de recherche dans TARGETS.
GET-TARGET	amène le mot suivant de TIB dans TARGETS en utilisant l'indice empilé comme déplacement.
.HEADER	mot de formatage d'en-tête dans le mode d'affichage SELECT.

SELECT	mot utilisateur permettant la sélection des champs à afficher.
STEP	mot utilisateur contrôlant l'affichage séquentiel des enregistrements.
CHAMPS	place les champs définis dans la liste de champs du fichier.
NEWFILE	initialise un nouveau fichier en mettant LASTREC et #ACTIVE à 0.
Q-ARRAYS	utilise le nombre empilé (qui doit être le nombre de conditions d'une recherche particulière) comme indices de boucle pour charger les tableaux de recherche avec les arguments trouvés dans TIB. La première entrée dans LOGICALS est toujours un NOOP.
FOUND?	compare les champs avec les arguments de recherche pour déterminer si les conditions (ou clefs) de recherche sont satisfaites.
FROM	exécute le mot suivant dans TIB, qui doit être un nom de fichier défini.
(FIND)	examine chaque enregistrement du fichier courant, et le compare (s'il n'est pas 'effacé') aux conditions spécifiées par l'interrogation.
CHERCHE	mot utilisateur d'interrogation (de recherche). Vérifie que le nombre d'arguments de la ligne de commande est correct, puis exécute (FIND)
NEXTREC	si le nombre d'enregistrements actifs est inférieur à LASTREC, le premier enregistrement 'effacé' (trouvé par FREE) est utilisé par la prochaine entrée. S'il n'en existe pas, le fichier est rallongé d'un enregistrement.
WRITE	parcourt la liste des champs du fichier courant, demandant et acceptant les entrées.
ENTRE	mot utilisateur générique d'entrée pour tous les champs définis par FICHIER.

Exemples d'utilisation.

Nous allons supposer que seuls les écrans 1 à 20 ont été compilés (la partie 'MENU' n'est pas disponible). C'est d'ailleurs l'utilisation prévue dans l'article d'origine. Après

NEWFILE JEDI , il faut entrer les enregistrements :

ENTRE JEDI , et le programme vous demande le premier champ du premier enregistrement:

SUJET : ______________.

et vous n'avez plus qu'à vous armer de patience...

Le mode STEP est pris par défaut. Pour rechercher les enregistrements des numéros 25 à 32 dont le sujet est FORTH ou LPB, entrons :

CHERCHE JEDI SUJET EST FORTH OU SUJET EST LPB ET NUMERO SUPRA 24 ET NUMERO INFRA 33 ,

et les enregistrements correspondant à ces clefs seront affichés un à un, avec possibilité de les corriger ou de les effacer.

En mode SELECT, il faut faire

SELECT JEDI SUJET TITRE NUMERO DATE , puis
CHERCHE JEDI SUJET EST LISP ET NUMERO SUPRA 15

et cette fois l'affichage des enregistrements trouvés se fait ligne par ligne , sous un en-tête des champs sélectionnés: SUJET, TITRE, NUMERO et DATE.

La compilation de la partie 'MENU' simplifie beaucoup ces manipulations, mais augmente évidemment la taille du programme.

----------------- Traduction/adaptation A. Jaccomard, Mai 1987.

0

```
                    FBASEII


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'    Ces écrans contiennent le programme  FBASEII de création '
' et recherche dans un fichier, ainsi qu'un fichier JEDI pour  '
' essais, qui est un index de la revue JEDI.                   '
'                                                              '
'    La base de ce programme est extraite d'un article paru    '
' dans FORTH DIMENSIONS, volume VIII/4, Nov/Dec 86 : "SIMPLE   '
' FILE QUERY" par E. Petsche.                                  '
'                                                              '
'   Pour utiliser ce programme et interroger le fichier, uti- '
' lisez la fonction 'AIDE' du MENU.                            '
'                                                              '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

1

```
\ FBASEII, chargement.                                28Mai87JaD

  ONLY FORTH ALSO DEFINITIONS

   1 18 +THRU
  20 LOAD  \ index pour JEDI.

CR 55 VDO .(  FBASEII est chargé et utilisable. ) 48 VDO
CR CR .(  Chargement et lancement du MENU d'utilisation.) CR

  20 22 +THRU
  24 32 +THRU
\ Le mot suivant doit être adapté à l'application, cf écr. 25.
: AUTO    EMPTY-BUFFERS  ONLY FORTH ALSO DEFINITIONS
     $OPEN   MENU ;
  ' AUTO IS BOOT  BOOT
```

2

```
\ DOER, MAKE, ;AND, UNDO : vectorisation.             28Mai87JaD

VARIABLE MARKER

: DOER    CREATE ['] NOOP  >BODY , DOES> @  >R ;

: (MAKE)  R>  DUP 2+  DUP 2+  SWAP @  >BODY ! @  ?DUP
   IF  >R  THEN ;

: MAKE   STATE @   IF  COMPILE (MAKE)  HERE MARKER !  0 ,
   ELSE  HERE  [COMPILE]  '  >BODY !  1 STATE !  INTERPRET
   THEN ; IMMEDIATE

: ;AND   COMPILE EXIT  HERE MARKER @  ! ; IMMEDIATE

: UNDO  ['] NOOP  >BODY  [COMPILE]  ' >BODY ! ;
```

3

```
\ Mots d'extension système.                           18Mai87JaD

: BLANK-PAD  PAD 80 BL FILL ;
: TEXT  ( c --)  BLANK-PAD  WORD COUNT PAD SWAP CMOVE> ;
: -TEXT ( ad1 n ad2 -- f )  2DUP + SWAP DO DROP 1+ DUP 1-
   C@  I  C@  - DUP  IF  DUP ABS / LEAVE THEN LOOP NIP ;
: -DOUBLE  ( a1 a2 -- f ) \ comme -TEXT pr nbr dbles.
   2@  ROT  2@  2SWAP  D-  2DUP  D0=
  IF  0  ELSE  2DUP  .0  D>  IF  1  ELSE  -1  THEN THEN
   >R  2DROP  R> ;
: ARRAY   CREATE  2* ALLOT  DOES>  SWAP 2* + ;
: IF-NOT  COMPILE  0=  [COMPILE]  IF ; IMMEDIATE
: WHILE-NOT  COMPILE 0=  [COMPILE] WHILE ; IMMEDIATE
```

4

```
\ Usage général.                                      29Mai87JaD
: ?OUI   (S -- f = t si 'O' ou 'o')
    KEY  SP@  1 UPPER  ASCII O = ;
: VDO    (S n -- )  \ mode vidéo.
   27 EMIT  91 EMIT  EMIT  109 EMIT ;
: CLS  27 EMIT  91 EMIT  50 EMIT 74 EMIT ; \ remplace DARK.
VARIABLE #HITS     \ nbr d'enrgats trouvés par la recherche.
VARIABLE FIN?      \ drapeau 'sortie de recherche' en mode STEP
VARIABLE IMPR      \ drapeau d'impression.
10 CONSTANT #AFF   \ nbr d'enrgats affichés à l'écran.
: IMPR?            \ bascule d'impression.
    IMPR @ IF  PRINTING ON  ELSE  PRINTING OFF  THEN ;
: SCRFUL?          \ arrêt sur écran plein.
   IMPR @ IF-NOT  #HITS @  #AFF MOD  IF-NOT
  CR 55 VDO ." Appuyez sur une touche" 48 VDO CR KEY DROP THEN
       THEN ;
```

5

```
\ Exploitation des paramètres du fichier.             28Mai87JaD
VARIABLE 'OPEN         \ pointe le bloc du fichier courant.
: REC#    'OPEN @ 8 + ;  \ adr du n° d'enregistrmt courant.
: LAYOUT  (S -- oct-enrgt, oct-bloc )
   'OPEN @ 4 + 2@ ;
: MAXRECS  ( -- n )  'OPEN @  2+ @ ;  \ nbr max d'enrgts
: READ     (S n -- )        \ fait de 'n' le n° d'enrgt courant.
   0 MAX DUP  MAXRECS  <  IF-NOT  ." numéro d'enregistrement inc
orrect "  QUIT  THEN    REC# ! ;
: RECORD  ( n -- a )        \ empile adr. du n-ième enrgmt.
   LAYOUT  */MOD  'OPEN @ @  + BLOCK + ;
: ADDRESS ( -- a )          \ empile adr. de l'enrgmt courant.
   REC# @ RECORD ;
: FIELD-LIST  ( -- a )  'OPEN @  10 + ;  \ adr liste des champs
: REC-LEN   'OPEN  @ 6 + @  ;           \ lgr d'un enrgt.
```

6

```
\ Mots "fichier" spécifiques.                         28Mai87JaD
: LASTREC   0 RECORD ;    \ n° dernier enrgt (lgr du fichier).
: #ACTIVE   0 RECORD 2+ ; \ nbr fich. non marqués par REMOVE.
: FICHIER   (S lg-enrgt,nbr-max-enrgt,n°-1er-blc -- )
   CREATE  ,   \ bloc de début du fichier.
     1+ ,       \ nbr max d'enrgmts du fich.
     DUP  B/BUF  OVER  /  *  ,  \ nbr d'oct/bloc.
     , 0  , 0 , \ oct/enrgmt, enrgmt courant et adr liste champs
   DOES>  'OPEN ! ;

VARIABLE 'FIELD   \ pointe le champ courant.

: CHAMP            \ utilisation : ALPHA 1 32 CHAMP TITRE.
   CREATE  , ( lgr )  , ( offset ) , ( type )
   DOES>  DUP 'FIELD !   2+ @  ADDRESS + ;
: FLD-WIDTH  ( -- n )  'FIELD @  @ ;    \ n = largeur du champ.
```

9

```
\ TABLES des fonctions dépendant du type.             28Mai87JaD

: T-COMPARE   ( a1 a2 -- n )  FLD-WIDTH SWAP -TEXT ;
: S-COMPARE   ( a1 a2 -- n )  SWAP @  SWAP @  - ;
: D-COMPARE   ( a1 a2 -- n )  -DOUBLE ;
TABLE COMPARE  T-COMPARE  S-COMPARE  D-COMPARE  D-COMPARE ;
: SUPRA     COMPARE 0> ;  \ pour "SUPERIEUR A"
: INFRA     COMPARE 0< ;  \ pour "INFERIEUR A"
: EST       COMPARE 0= ;
: NESTPAS  COMPARE     ;
\ Mots d'affichage d'enrgmts.
: .FIELD-NAME IMPR? 'FIELD @  BODY> >NAME  .ID  ASCII : EMIT ;
: .FIELD IMPR?  DISPLAY ;
: .LINE  IMPR?  CR  .FIELD-NAME  SPACE  DISPLAY ;
: .RECORD  FIELD-LIST @  BEGIN  DUP @ ?DUP  WHILE  EXECUTE
   .LINE  2+  REPEAT  DROP ;
```

7

```
\ TABLES des fonctions dépendant du type.          28Mai87JaD

0 CONSTANT ALPHA    \ déplacement dans les tables de fonctions.
2 CONSTANT SIMPLE
4 CONSTANT DOUBLE
6 CONSTANT $$
: NOMBRE   NUMBER?  IF-NOT  2DROP
   49 VDO  CR CR  ." Entrez un nombre ... "  48 VDO
   500 MS BOOT THEN ;
: FLD-TYPE  ( -- n )  'FIELD @  4 + @ ;  \ type du champ.
: TABLE   :  DOES> FLD-TYPE + @ EXECUTE ;
: PAD>NUM   ( -- d )  PAD 1-  FLD-WIDTH OVER  C! NOMBRE ;
: T-!   ( adr -- )  PAD  SWAP  FLD-WIDTH CMOVE ;
: S-!   ( adr -- )  PAD>NUM  DROP SWAP ! ;
: D-!   ( adr -- )  PAD>NUM  ROT 2! ;
TABLE (ENTER)  T-!  S-!  D-!  D-! ; \ place entrées de champ.
```

8

```
\ TABLES des fonctions dépendant du type.          28Mai87JaD

: $FORMAT  ( d -- adr u )  DUP >R DABS <# # # ASCII . HOLD #S
   R> SIGN #> ;     \ affichage 'd' sous format $ddd.cc.

: T-TYPE   ( adr -- )  FLD-WIDTH  TYPE ;
: S-TYPE   ( adr -- )  @  FLD-WIDTH  .R ;
: D-TYPE   ( adr -- )  2@  FLD-WIDTH  D.R ;
: $-TYPE   ( adr -- )  2@  $FORMAT  FLD-WIDTH  DUP ROT -
   SPACES  TYPE ;

TABLE DISPLAY  T-TYPE   S-TYPE   D-TYPE   $-TYPE ;
```

10

```
\ Mots d'entrées.                                  28Mai87JaD
: DASHES  ( n -- )  SPACE DUP 0 DO 95 EMIT LOOP
  0 DO 8 EMIT LOOP ;  \ utilisé dans le "prompt" d'entrée.
: INPUT   QUERY  ASCII &  TEXT ;
: .PROMPT  CR  CR  .FIELD-NAME  SPACE  FLD-WIDTH DASHES ;
: ENTER \ prompte, accepte et place les entrées des champs
   .PROMPT  INPUT  (ENTER)  UPDATE ;
\ Mots de recherche.
: REMOVED?  ( rec# -- ? )  RECORD   C@  ASCII * = ;
: #ARGS  ( -- n )   \ compte les arguments de la ligne de cdes
   >IN @   0  BEGIN  BL WORD  C@  WHILE  1+  REPEAT  SWAP
   >IN ! ;
DOER   .DISPLAY     \ vectorisation.
DOER   DELAY
DOER   HEADING
DOER   AFFREC
```

11

```
\ Mots de modification de fichier.                 18Mai87JaD

: SIGNAL 7 EMIT CR COUNT TYPE
  55 VDO  ."  n'est pas un champ valide."  48 VDO ;

: CHANGE  BEGIN  CR
 55 VDO ." Entrez le nom du champ à modifier." 48 VDO CR QUERY
  BL WORD FIND  WHILE-NOT  SIGNAL  REPEAT  EXECUTE  ENTER ;

: EFFACE
  ASCII *  ADDRESS  C!  UPDATE  -1 #ACTIVE +!  UPDATE ;

: MODIFIE  CR  55 VDO
." Entrez C pour changer ou E pour effacer un enregistrement."
  48 VDO  KEY SP@ 1 UPPER DUP ASCII C = IF  DROP CHANGE
  ELSE  ASCII E = IF  EFFACE  THEN  THEN  CR  DELAY ;
```

12

```
\ Mots de conditions de recherche.                 25Mai87JaD

4 CONSTANT Q#   \ nbr max de conditions de recherche.

Q# ARRAY LOGICALS
Q# ARRAY OPERANDS
Q# ARRAY CONDITIONS

: TARGETS   HERE 200 + ;
: +TARGET   ( i -- )  30 * TARGETS + ;
: T-BRING   ( a -- )  TEXT  PAD SWAP FLD-WIDTH CMOVE ;
: 1-BRING   ( a -- )  WORD NOMBRE  DROP SWAP ! ;
: 2-BRING   ( a -- )  WORD NOMBRE  ROT 2! ;
TABLE BRING  T-BRING  1-BRING  2-BRING  2-BRING ;
: GET-TARGET  ( i -- )  +TARGET  BL  BRING ;
```

13

```
\ Mots de gestion.                                 18Mai87JaD

: Q-ARRAYS  ( n -- )
   ['] NOOP  0 LOGICALS !
   0  DO  I  IF  '  I LOGICALS !  THEN
   '  DUP I OPERANDS !  >BODY 'FIELD !
   '  I  CONDITIONS !  I GET-TARGET  LOOP ;

: LOGIC      ( i -- )  LOGICALS   PERFORM ;

: OPERAND    ( i -- )  OPERANDS   PERFORM ;

: CONDITION  ( i -- )  CONDITIONS PERFORM ;

: TARGET   +TARGET ;
```

15

```
\ Mots de gestion.                                 28Mai87JaD

: CHERCHE  \ mot de gestion "utilisateur".
  FROM
  #ARGS 1+ Q# /MOD SWAP
    IF ." Trop d'arguments de recherche." 500 MS BOOT  THEN
  CR  HEADING  DUP  Q-ARRAYS  (FIND) ;
: ET AND ;
: OU OR ;
\ utilisation: CHERCHE JEDI SUJET EST LISP ET NUMERO SUPRA 15

: DONE?  ( -- f: vrai si pas d'autres entrées )
   CR  55 VDO ." autre entrée ? (O/N) "  48 VDO   ?OUI NOT ;
```

16

```
\ Mots d'entrée dans le fichier.                   22Mai87JaD
: NEWFILE   FROM  0 #ACTIVE !   0 LASTREC !  ;
: FREE  ( -- rec# )  LASTREC @  1+ DUP 1  DO  I REMOVED?
    IF DROP I LEAVE THEN  LOOP ;
: NEXTREC  ( -- )   LASTREC @  #ACTIVE @  >
    IF  FREE  REC# !  ADDRESS REC-LEN BL FILL  UPDATE
  ELSE  LASTREC  DUP @ 1+ DUP READ  SWAP !  UPDATE  THEN ;
: WRITE
   FIELD-LIST @  BEGIN  DUP @ ?DUP  WHILE  EXECUTE  ENTER 2+
   REPEAT  DROP ;

: ENTRE  #ARGS  1 <>
    IF ." Il me faut un nom de fichier ..." 500 MS BOOT  THEN
  FROM  BEGIN  CLS  CR CR CR  NEXTREC  WRITE  1 #ACTIVE +!
  UPDATE  CR CR CR  DONE?  UNTIL
  SAVE-BUFFERS ;
```

```
14
\ Mots de gestion.                                   25Mai87JaD

: FOUND?  ( n -- f )
   0  DO  I OPERAND  I TARGET  I CONDITION   I LOGIC  LOOP ;
: FROM                  \ FROM <nom_de_fichier>
    '    EXECUTE ;
: (FIND)  ( n -- )
   0 #HITS !   LASTREC @ 1+  1
  DO  I REMOVED?
    IF-NOT  I REC# !   ( n )  DUP  FOUND?
        IF  1 #HITS +! CR  .DISPLAY  CR DELAY
 FIN? @  IF  FIN? OFF  LEAVE  THEN  AFFREC  \ arrêt 'écr. plein'
   THEN  THEN
  LOOP  DROP  #HITS @  IF-NOT
   CR  49 VDO ." Je n'ai pas trouvé..." 48 VDO 500 MS THEN ;
```

```
17
\ Affichage.                                          18Mai87JaD

VARIABLE EXTRAITS  12 ALLOT  \ pointe sur le champ à afficher.

: DASH-LINE  CR  72  0  DO  ASCII - EMIT  LOOP  CR ;

: .HEADER    IMPR?
   EXTRAITS  BEGIN  DUP @  ?DUP  WHILE  DUP  >BODY
   'FIELD !  BODY>  >NAME  DUP  .ID   C@ 31 AND  FLD-WIDTH
   SWAP  -  ABS  1+ SPACES  2+  REPEAT  DROP  DASH-LINE  CR ;

: SPREAD   FLD-TYPE  IF  'FIELD @  BODY>  >NAME  C@  31 AND
   FLD-WIDTH SWAP - ABS  FLD-WIDTH + SPACES  ELSE  2 SPACES
   THEN ;
```

```
18
\ Mots d'affichage de fichier.                        28Mai87JaD

: .EXTRAITS
  EXTRAITS  BEGIN  DUP @  ?DUP  WHILE  EXECUTE
   .FIELD  SPREAD  2+ REPEAT DROP ;
: SELECT  \ utilisation: SELECT <nom_fich> <champ1> ... <champN>
  FROM  EXTRAITS  #ARGS  DUP  5 >
   IF ." Trop d'arguments... " 500 MS BOOT  THEN
 0  DO  '  OVER !  2+ LOOP  0 SWAP !
      MAKE  DELAY    NOOP
   ;AND MAKE  AFFREC   SCRFUL?
   ;AND MAKE .DISPLAY .EXTRAITS
   ;AND MAKE  HEADING .HEADER ;
: CHAMPS  \ utilisation:<nom_fich> N CHAMPS <champ1> <champ2>...
  HERE  SWAP  0  DO  '  ,  LOOP  0  ,  FIELD-LIST ! ;
```

```
19
\ Mots d'affichage de fichier.                        29Mai87JaD

: .MSSG
  PRINTING OFF
  CR 55 VDO ."  ENTER pour sortir    ESC pour modifier "
   ."    Touche quelconque pour continuer " 48 VDO CR ;

: STEP
       MAKE  DELAY   .MSSG
  KEY  DUP 27 =  IF  DROP  MODIFIE
  ELSE  13 =  IF  CR 55 VDO ."  FIN de GESTION. " 48 VDO CR
    SAVE-BUFFERS FIN? ON  EXIT  THEN THEN
  ;AND MAKE .DISPLAY .RECORD
  ;AND MAKE  AFFREC   NOOP
  ;AND MAKE  HEADING  NOOP ;
   STEP    \  affichage par défaut.
```

```
20
\ Application: index pour JEDI.                       29Mai87JaD

  100 500 40 FICHIER JEDI   \ lg_enrg, nbr_max_enrg, n°_1er_blc.

 ALPHA   0  6  CHAMP  SUJET  \ type,1er_oct,lgr,,nom_du_champ
 ALPHA   6 32  CHAMP  TITRE
 SIMPLE 38  2  CHAMP  NUMERO
 SIMPLE 40  2  CHAMP  PAGE
 ALPHA  42  5  CHAMP  DATE
 ALPHA  47 40  CHAMP  REM

 JEDI  6 CHAMPS SUJET TITRE NUMERO PAGE DATE REM

\ NEWFILE JEDI
\S Ce mot ne doit être exécuté que lors de la première compilati
on du nouveau fichier, mais il doit alors l'être impérativement.
```

```
21
\ APPUI, SAVCUR, ...                                  20Mai87JaD

: APPUI
  CR CR 20 SPACES  55 VDO ." Appuyez sur une touche pour revenir
au menu." 48 VDO  KEY DROP ;

CREATE $FICH 12 ALLOT

: .FICH     \ affiche nom du fichier courant.
    0 1 AT  55 VDO  $FICH COUNT TYPE  48 VDO ;

: +$FICH  (S adr lgr --- ) \ ajoute $FICH à la chaîne spécifiée
  2DUP  $FICH COUNT 2SWAP 12 - + SWAP CMOVE ;
```

```
22
\ OUVRIR.                                             18Mai87JaD

: OUVRIR   CLS
  5  5 AT ." Quel fichier voulez-vous consulter ?"
 10  8 AT ." 1 - JEDI."
 10 10 AT ." 2 - FORTH DIMENSIONS."
 10 12 AT ." 3 - REVUE UTILISATEUR IBM-PC."
 20 15 AT 55 VDO ." Votre choix ?"  48 VDO   KEY  48 -
CASE    1 =: " JEDI "    $FICH  PLACE ;;
        2 =: " F-DIM "   $FICH  PLACE ;;
        3 =: " IBM-REV " $FICH  PLACE ;;
  NOCASE =: RECURSE ;;
CASEND ;
```

```
23
\ AIDE.                                               22Mai87JaD
: AIDE    CLS  .FICH
  49 VDO 20 1 AT ." A I D E S" 48 VDO
 5 5 AT ." 1 - Un fichier doit être ouvert préalablement à toute
opération."
 5 6 AT ." 2 - La recherche d'un/des enregistrement(s) se fait e
n donnant une ou des  clef(s) de recherche, 4 au maximum pour ce
 programme. Les clefs utilisables sontindiquées lors de l'appel
des fonctions 'Recherche Normale' et 'Recherche Sélec-tive'." CR
 5 SPACES ." 3 - En Recherche Normale, tous les enregistrements
trouvés sont affichés   l'un après l'autre." CR
 5 SPACES ." 4 - En Recherche Sélective, seules les champs désig
nés sont affichés." CR
 5 SPACES ." 5 - Le résultat des recherche peut être envoyé à l'
imprimante : la fonction 5 est une bascule." CR  -->
```

24

```
\ AIDE, suite.                                          22Mai87JaD
 5 SPACES ." 6 - Les modifications sur un enregistrement se font
en Recherche Normale." CR
 5 SPACES ." 7 - Extension du fichier : fonction 4."  CR
 5 SPACES ." 8 - En cas d'erreur (sortie du programme), rappelez
celui-ci par 'MENU'." ;
```

25

```
\ ENTREE.                                               29Mai87JaD

: $EXEC   (S adr lg -- )    \ exécute la commande 'chaînée'.
     DUP  #TIB !  TIB SWAP CMOVE  BLK OFF  >IN OFF  INTERPRET ;
: $ENTRE   (S -- adr lg )  \ crée la chaîne ENTRE <nom-fichier>
    " ENTRE              " +$FICH ;

: $OPEN     \ ouvre le fichier 'application' : METTEZ SON NOM ICI
    " OPEN B:FBASEII.BLK"  $EXEC ;

: .CHAMPS   \ affiche les noms des champs du fichier courant.
   FIELD-LIST  @   ( adr_liste_champs )
  BEGIN   DUP @  ?DUP  WHILE
   >NAME  DUP 1+ SWAP C@ 31 AND           (S -- adr lg )
     BOUNDS DO  I  C@ 127 AND EMIT LOOP  \ supprime bit 7.
   CR CR 10 SPACES  2+  REPEAT  DROP ;
```

26

```
\ RECHERCHE.                                            28Mai87JaD

: $CHERCHE   (S -- adr lg )
    " CHERCHE              " +$FICH ;

: $INTERP    \ interprète commande entrée au clavier.
     TIB #TIB @ + 80 EXPECT SPAN @ #TIB +!
     TIB #TIB @  $EXEC ;

: RECH          \ recherche selon critères donnés.
     >IN OFF  $CHERCHE  DUP #TIB !  TIB  SWAP  CMOVE
                \ place 'CHERCHE XXXXX' début TIB.
     CR CR  ." Entrez les 'clefs' de recherche : " CR CR
     $INTERP  ;
```

30

```
\ SORTIE.                                               18Mai87JaD

: SORTIE           \ retour sous FORTH.
  CLS  5 5 AT 55 VDO ."  Etes-vous sûr ? (O/N) " 48 VDO  ?OUI
 IF  5 8 AT 55 VDO ." Sortie de FBASEII et retour sous FORTH"
 48 VDO CR CR  QUIT  THEN ;

: DOS?             \ retour sous DOS.
  CLS 55 VDO 10 10 AT ."  Etes-vous sûr ? " 48 VDO   ?OUI
  IF CR 49 VDO ." ON SORT" 48 VDO BYE THEN ;
```

27

```
\ RECHERCHE.                                            18Mai87JaD

: $SELECT   (S -- adr lg )
    " SELECT             " +$FICH ;

: RECH-SEL       \ recherche sélective.
   CLS  5 2 AT ." Seuls sont affichés les 'champs' que vous choi
sissez : "  10  5 AT .CHAMPS
    >IN OFF  $SELECT  DUP #TIB !  TIB SWAP CMOVE
    CR CR  ." Entrez les 'clefs' de sélection :" CR CR
    $INTERP  RECH ;
```

28

```
\ RECHERCHE.                                            29Mai87JaD

: RECH-NORM
  STEP  CLS  5 2 AT ." La recherche se fait selon les critères,
ou clefs : "  10 5  AT .CHAMPS CR CR
 5 SPACES ." Vous pouvez en utiliser au maximum 4 simultanément,
séparées par l'un des  'filtres' : EST, NESTPAS, ET, OU, SUPRA,
INFRA."
 CR CR   ." Ex.: SUJET EST FORTH ET NUMERO SUPRA 25"
 RECH ;
```

29

```
\ Modification/Suppression d'un enregistrement.        29Mai87JaD

: MODSUP            \ 'mode d'emploi'.
   CLS
  5 5 AT ." Pour modifier ou supprimer un enregistrement (une fi
che), utilisez l'option 'Recherche normale' du MENU."  CR
 5 SPACES ." Lorsque vous avez trouvé l'enregistrement, utilisez
les touches 'ESC' et  'C' ou 'E' selon le cas."  CR CR
 5 SPACES ." Pour ajouter  un/des  enregistrement(s)  au  fichie
r ouvert, utilisez  la  fonction 'Extension du fichier' du MENU.
"   APPUI ;
```

33

```
\ MENU                                                  28Mai87JaD

: MENU                         \ MENU général, lancé par BOOT.
    PRINTING OFF   CHOIX   KEY  48 -
CASE  1 =: OUVRIR ;;
      2 =: RECH-NORM ;;
      3 =: RECH-SEL  APPUI ;;
      4 =: $ENTRE  $EXEC ;;
      5 =: MODSUP  ;;
      6 =: IMPRIME ;;
      7 =: AIDE  APPUI ;;
      9 =: SORTIE ;;
      0 =: DOS?   ;;
 CASEND   RECURSIVE MENU ;
```

31

```
\ Impression.                                   18Mai87JaD

: IMPRIME               \ bascule d'impression.
  CLS  49 VDO 10 10 AT
  INPR @
   IF  INPR OFF   ." Pas d'impression"
  ELSE  INPR ON   ." Impression"
   THEN ."  des résultats de la recherche."
   APPUI ;
```

32

```
\ CHOIX                                         16Mai87JaD

: CHOIX  CLS  .FICH
 20 1  AT 49 VDO ." M E N U " 48 VDO
   10  5 AT ." 1 - Ouverture d'un fichier."
   10  7 AT ." 2 - Recherche normale."
   10  9 AT ." 3 - Recherche sélective."
   10 11 AT ." 4 - Extension du fichier."
   10 13 AT ." 5 - Suppression d'un enregistrement."
   10 15 AT ." 6 - Impression des résultats."
   10 17 AT ." 7 - AIDE."
49 VDO 15 19 AT ." 9 - Retour sous FORTH."
       15 21 AT ." 0 - Sortie vers DOS."   48 VDO
 20 23 AT 55 VDO ." Votre choix ? " 48 VDO ;
```

34

```
NOTES sur la vectorisation.                     29Mai87JaD

      Les mots d'exécution vectorisée DOER et MAKE de
l'écran #2 sont tirés de THINKING FORTH par L. BRODIE. Ils
sont utilisés ici, au lieu des DEFER et IS de F83 car ils
proposent une méthode différente et parfois subtile (voir
la définition de STEP de l'écran #19) de vectorisation.

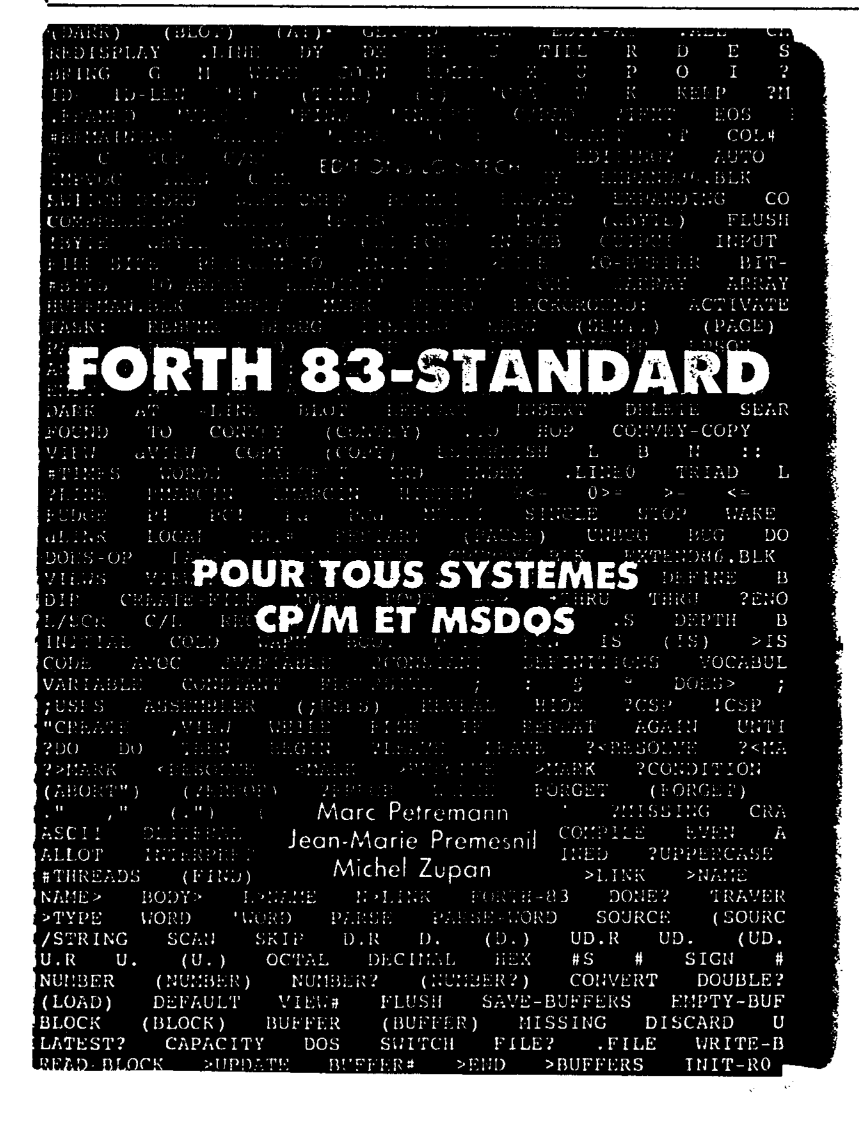
DOER <mot>  à la compilation, crée l'entête <mot> et place dans
            son PFA le CFA de NOOP;
           à l'exécution, place sur RP l'adresse contenue dans
            le CFA de <mot>.
MAKE <mot_1> <mot_2>  vectorise <mot_1> sur <mot_2> (c. à d.
            exécute <mot_2> à l'appel de <mot_1>).
     Ces mots peuvent être remplacés par DEFER et IS, dans les
écrans 10, 18 et 19.
```

35

```
NOTES sur la syntaxe des mots 'utilisateur'.    29Mai87JaD

CREATION d'un fichier (cf écr. #20):
 lg-enr,nbr-max,1er-blc FICHIER <nom-fich>
 type,1er-oct,lg CHAMP  <champ1>
 type,1er-oct,lg CHAMP  <champ2>
      ...........
 <nom-fich>  nbr_champs  CHAMPS  <champ_1> <champ_2> ...

UTILISATION :
Mode STEP    (4 arguments de recherche au maximum)
  CHERCHE <nom_fich> <champ_X> [EST,NESTPAS] <valeur>
       [ET,OU] <champ_Y> [EST,NESTPAS,SUPRA,INFRA] <valeur> ...
Mode SELECT  (5 champs affichés au maximum)
  SELECT <nom_fich> <champ_1> <champ_2> ... <champ_N>
   puis comme pour STEP.
```



VARIABLES CHAINES EXECUTABLES

par Michel ZUPAN

La présente étude tente de répondre à quelques unes des propositions emises par Marc PETREMANN dans le numéro 33 de JEDI concernant la génération de commandes FORTH à partir de chaines de caractères.

L'ensemble est developpe en F83 pour des raisons qu'on aura aucun mal à saisir au vu des nombreux mots et concepts pris dans ce standard. Je crains même que sa transposition pour d'anciens FORTH, toujours possible bien sûr, ne se révèle parfois délicate.

Reprenons le principe de l'exécution (interprétation serait un terme plus juste) d'une chaine explicite FORTH.

Soit la constante chaine suivante:

: TEST " DARK WORDS " ;
TEST TYPE affiche **DARK WORDS**
TEST EXECUTE$ efface l'écran et liste les mots du vocabulaire courant.

Si nous gardons la premire définition de M.P. rappelée dans le premier écran et qui consiste à déplacer la chaine dans le TIB (tampon du terminal d'entrée), le contenu de celui-ci est modifié, ce qui est assez gênant.

Ainsi, la ligne suivante:

TEST EXECUTE$ CR .(l'ai-je bien descendu?)

ne fonctionne pas correctement. De même, la commande ne peut être placée dans un écran sans interrompre son interprétation.

En F83, la solution consiste à détourner provisoirement la SOURCE (adresse et longueur) à interpréter vers notre chaine, laquelle sera délivrée par une pseudo-constante chaine **STR$**. Une fois celle-ci interprétée, on restitue la source d'origine avec son pointeur d'interprétation.

C'est ce que propose la seconde version de **EXECUTE$** qui corrige les inconvénients de la première: le TIB n'est plus modifié.

Malheureusement, cette seconde version se révèle bien plus dangeureuse que la première! Essayez:

: TEST2 " N'IMPORTE QUOI! " ;
TEST2 EXECUTE$

Vous pouvez éteindre la machine! L'interpréteur est bloqué dans **TEST2** et rien ne peut l'en faire sortir.

D'où la règle absolue valable pour toutes les procédures de **"méta-interprétation"**: ne jamais déplacer l'interpréteur sur une autre source sans une gestion adéquate des erreurs pouvant y survenir.

La procédure est en fait assez simple: outre SOURCE, il faut détourner **?ERROR** pour que soit restituée en cas d'erreur la source d'origine. On utilisera également une variable pour conserver et restituer le pointeur **>IN** d'interprétation.

Le second écran donne une troisième version de **EXECUTE$** enfin acceptable. Méfiez-vous toutefois encore des re-entrances d'interprétation (oh que c'est vicieux ça) comme de placr un **QUIT** à la fin de **TEST** (essayez, c'est assez drole): le F83 n'a pas tout prévu!.

Passons aux variables chaines qui peuvent désormais être considérées comme autant de tampons d'interprétation (entre autres applications):

2 7 THRU compile l'ensemble des outils proposés.

" (quote) est redéfini pour fonctionner aussi bien en compilation qu'en interprétation, n'en déplaise au F83 qui semble ne guere priser ce genre de mixité (voir **."** et .( ).

STRING définit une variable chaine selon le cahier des charges de M.P.

80 STRING A$

crée un tampon de 80 caractres maximum, vide à l'initialisation et qui peut être rempli par l'opérateur d'affectation **$!** :

" GO FORTH" A$ $!

LEN$ délivre la longueur maximale d'une variable chaine.

INPUT$ affecte une variable chaine depuis le terminal.

RIGHT$**, **LEFT$, **MID$** sont aisés à reconnaitre.

A$ 5 RIGHT$ TYPE affiche **FORTH**
A$ 2 LEFT$ TYPE affiche **GO**
A$ 3 4 MID$ TYPE affiche **FORT**

ITEM isole un mot séparé par des espaces (Ndlr: ce mot est à rapprocher du fonctionnement de ITEM en LOGO...):

A$ 2 ITEM EXECUTE$ exécute **FORTH**

INSERT$ insere une chaine dans une variable:

" OO" A$ 2 INSERT$ A$ TYPE affiche **GOOD FORTH**

APPEND$ rest l'opérateur de concaténation:

" AND MULTIPLY !" A$ APPEND

ERASE$ efface une portion de chaine:

A$ 2 2 ERASE$
A$ TYPE affiche **GO FORTH AND MULTIPLY !**

SEARCH$ cherche la premire occurence d'une sous-chaine

" AND " A$ SEARCH$
DROP A$ ROT LEFT$ TYPE affiche **GO FORTH**

DELETE$** complète **ERASE$ en effaçant une sous-chaine dans une variable:

A$ 2 ITEM A$ DELETE$
A$ TYPE affiche **GO AND MULTIPLY !**

Reste à expliquer à l'interpréteur FORTH comment exécuter cette sentence biblique!

```
A-STRINGVAR.BLK / SCRN# 1
( 0) \ interprétation et exécution d'une chaine            25avr87 MZ
( 1) ONLY FORTH DEFINITIONS DECIMAL
( 2) : EXECUTE$ ( adr len --- )          \ 1ère version
( 3)             DUP #TIB !              \ -----------
( 4)             TIB SWAP MOVE           \ déplace la chaine dans TIB
( 5)             BLK OFF >IN OFF ;
( 6)
( 7) 0 0 2CONSTANT STR$
( 8) : EXECUTE$ ( adr len --- )          \ 2ème version
( 9)             ['] STR$ >BODY 2!       \ -----------
(10)             ['] STR$    IS SOURCE   \ vectorise SOURCE sur STR$
(11)             >IN >R >IN OFF
(12)             RUN
(13)             ['] (SOURCE) IS SOURCE  \ rétablit SOURCE et >IN
(14)             R> >IN ! ;
(15)
```

```
A-STRINGVAR.BLK / SCRN# 2
\ Interprétation et exécution d'une chaîne            25avr87 MZ
0 0 2CONSTANT STR$  VARIABLE (>IN)
: ?ERROR$  ( adr len flag --- )    \ gestion des erreurs dans
           DUP IF                  \ une SOURCE temporaire :
           ['] (SOURCE) IS SOURCE  \ restitue SOURCE d'origine
           ['] (?ERROR) IS ?ERROR
           (>IN) @ >IN ! THEN (?ERROR) ;
: EXECUTE$ ( adr len --- )         \ 3ème version
           ['] STR$ >BODY 2!       \ -----------
           ['] STR$     IS SOURCE
           ['] ?ERROR$  IS ?ERROR  \ avec gestion d'erreurs
           >IN @ (>IN) ! >IN OFF
           RUN
           ['] (?ERROR) IS ?ERROR
           ['] (SOURCE) IS SOURCE
           (>IN) @ >IN ! ;
```

```
A-STRINGVAR.BLK / SCRN# 3
\ Variables chaînes                                   25avr87 MZ

: "      ( <text"> --- adr len )
         STATE @ IF COMPILE (") ,"    \ en compilation
                 ELSE ASCII " PARSE   \ en exécution
                 THEN ; IMMEDIATE

: STRING ( <var$name> lenmax --- )    \ mot de définition d'une
         ( --- adr len )              \ variable$ chaîne
         CREATE DUP C, 0 C, ALLOT     \ de longueur maximale
         DOES> 1+ COUNT ;             \ lenmax

: $!     ( str strvar --- )           \ affectation sécurisée
         DROP 1- DUP 1- C@ ROT MIN    \ d'une variable chaîne
         SWAP PLACE ;
```

```
A-STRINGVAR.BLK / SCRN# 4
\ Principales fonctions sur chaînes                   25avr87 MZ
: LEN$    ( strvar --- strvar lenmax ) \ longueur d'une variable$
          OVER 2- C@ ;                 \ de type chaîne

: INPUT$ ( strvar --- )                \ entrée au terminal
         OVER >R LEN$ NIP EXPECT       \ d'une variable$ chaîne
         SPAN @ R> 1- C! ;
                                       \ fonctions sécurisées :
: RIGHT$ ( str1 len --- str2 )         \ partie droite
         0 MAX OVER MIN >R + R@ - R> ;

: LEFT$  ( str1 len --- str2 )         \ partie gauche
         0 MAX MIN ;

: MID$   ( str1 pos len --- str2 )     \ sous-chaîne
         >R OVER SWAP - RIGHT$ R> LEFT$ ;
```

```
A-STRINGVAR.BLK / SCRN# 5
\ Extraction d'un mot d'une chaîne                    25avr87 MZ

: ITEM ( str1 n --- str2 )            \ isole le n-ième 'mot'
      1 MAX >R TUCK R>                \ séparé par des espaces
      0 DO ROT DROP                   \ dans une chaîne
           BL SKIP OVER SWAP
           BL SCAN                    \ laisse une chaine vide
      LOOP                            \ si hors limites
      DROP OVER - ;
```

```
A-STRINGVAR.BLK / SCRN# 6
\ Operateurs sur variables chaînes                    25avr87 MZ
: INSERT$ ( str strvar pos --- )      \ inserre une chaîne dans
          0 MAX OVER MIN              \ une variable$ à la
          -ROT LEN$ SWAP              \ position pos
          4 PICK + OVER MIN
          2 PICK 1- C!
          2 PICK - >R + R> INSERT ;

: APPEND$ ( str strvar --- )          \ ajoute une chaîne à la
          DUP INSERT$ ;               \ fin d'une variable$
```

```
A-STRINGVAR.BLK / SCRN# 7
\ Opérateurs sur variables chaînes                    25avr87 MZ
: ERASE$  ( strvar pos len --- )              \ détruit len caract.
          >R 0 MAX OVER MIN                    \ à la position pos
          2DUP - R> MIN 0 MAX                  \ dans une variable$
          2 PICK OVER - 4 PICK 1- C!
          ROT OVER - 2 PICK -
          >R -ROT + TUCK + SWAP
          R> CMOVE ;

: SEARCH$ ( str strvar --- pos len )           \ cherche une chaîne
          2 PICK >R SEARCH                     \ dans une variable$
          IF R> ELSE R> 2DROP 0 0 THEN ;

: DELETE$ ( str strvar --- )                   \ détruit une chaîne
          2DUP >R >R SEARCH$                   \ dans une variable$
          R> R> 2SWAP ERASE$ ;
```

APL

RECREATIONS APL
LE PGCD
par F. ESPINASSE

L'APL est un langage récursif, c'est à dire qu'une fonction peut s'appeler elle-même. Pour illustrer la récursivité d'un langage, un exemple classique est l'écriture d'une fonction factorielle. Malheureusement (ou heureusement) la factorielle est en APL un des opérateurs de base, et il ne parait pas profitable d'écrire un fonction qui réalise ce qu'on peut programmer par un simple point d'exclamation ! (de même pour l'inversion de matrice qui se programme par un symbole domino). Il ne nous reste plus qu'à nous rabattre sur les tours de Hanoï ou sur le PGCD.

Le PGCD (Plus Grand Commun Diviseur) de deux nombres entiers, X1 et X2 peut se calculer par l'algorithme d'Euclide, qui s'appuie sur la remarque suivante: si l'on suppose X1 ≤ X2, il y a deux cas possibles:

1) X1 divise X2 sans reste, dans ce cas, il est le nombre cherché; on s'arrête là.

2) X1 ne divise pas X2, dans ce cas, le PGCD de X1 et X2 sera trouvé en recherchant le PGCD de X1 et de (X2 modulo X1); d'où la récursivité.

X2 modulo X1 est le reste de la division de X2 par X1 et s'écrit en APL:

X1 | X2

La fonction PGCD pourra s'écrire:

```
Z←PGCD X;A
⍝ Calcul du  Plus Grand Commun Diviseur de X[1] et X[2]
⍝ ------------------------------------------------------
⍝ soit Z le plus petit. et A le plus grand des 2 nombres
Z←⌊/X
A←⌈/X
⍝ Remarque: il se peut que A=Z
⍝ si Z divise A(A modulo Z est nul) le resultat cherche est Z :STOP
→(0=Z|A)/0
⍝ sinon le resultat est le PGCD de Z et de A modulo Z
Z←PGCD Z,Z|A
```

Nous ne la détaillerons pas, car elle est parfaitement lisible (les commentaires ne sont pas écrits en japonais). Mais nous ferons cependant quelques remarques:

- le paramètre d'entrée X doit être un vecteur à deux éléments, c'est à dire que l'appel de la fonction devra comporter le nom de celle-ci suivi de deux nombres entiers, séparés par une virgule ou par un espace. Exemple:

PGCD 30 40
10

- l'en-tête de la fonction comprend le nom de la variable **A** précédé d'un point-virgule. Celà veut dire que la variable A est **locale**. Elle n'a d'existence que pendant la période d'activation de la fonction. Elle ne peut être accédée par une fonction appelante. Celà a pour effet de ne pas mélanger enmémoire une variable d'une fonction appelée avec la variable homonyme de la fonction appelante (Ndlr: c'est le cas d'à peu près tous les langages, excepté BASIC... quoique les dernières versions, style QuickBASIC...). Il est sage de **localiser** systématiquement toutes les variables d'une fonction, sauf besoin particulier. Une variable **globale** (non locale) est assimilable à un **COMMON** de certains langages.

- en ce qui concerne le passage de paramètre(s) entre fonction appelante et appelée, il faut savoir que la fonction appelée ne travaille pas sur la valeur en mémoire du paramètre, mais sur une copie. Il n'y a donc pas d'effets de bord en APL.

Pour nous distraire encore un peu, nous pourrons écrire une fonction PPCM pour calculer le Plus Petit Commun Multiple de deux nombres. Pour celà, nous écrirons que le PPCM de deux nombres est égal à leur produit divisé par leur PGCD

```
Z←PPCM X
⍝ Calcul du Plus Petit Commun Multiple de X[1] et X[2]
⍝ ----------------------------------------------------
Z←(×/X)÷PGCD X
```

Nous attequerons les Tours de Hanoï avec le prochain Retour du Jedi (Ndlr: attention, il vous attend avec le même sujet en FORTH au détour).

■

INDEX THEMATIQUE

Cet index reprend le contenu des numéros de la revue JEDI 7 à 35. En face de chaque rubrique figure entre parenthèse le numéro de la revue où est paru l'article concerné.

TARIF DES REVUES

N°s 7 à 14, 10 Fr pièce
N°s 15 à 17, 15 Fr pièce
N°s 18 à 35, 20 Fr pièce